Scalable and Lightweight Cloud-Native Application Sandboxing

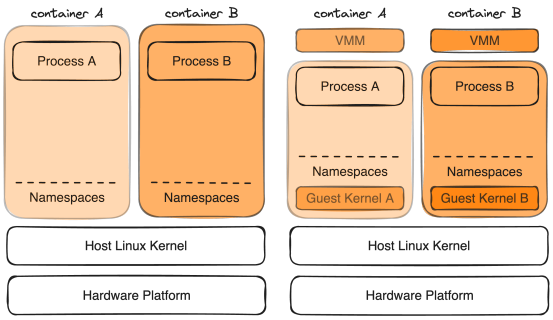
***Abstract*—In the evolving landscape of edge-cloud systems, the need for efficient and secure application deployment is critical. This work introduces a scalable and lightweight approach to cloud-native application sandboxing using urunc, a container runtime for unikernels, and bima, a tool for packaging unikernels into OCI-compatible artifacts. By leveraging both software- and hardware-based sandboxing, our solution enhances the security and efficiency of edge-cloud applications while maintaining compatibility with existing cloud-native ecosystems. Our contributions also extend to open-source projects like kata-containers, enabling AWS Firecracker microVM sandboxing, and integrating vAccel, to expose hardware acceleration functionality to sandboxed workloads.**

***Index Terms*—lightweight virtualization, isolation, security, 6G, NFV, hardware acceleration, vAccel**

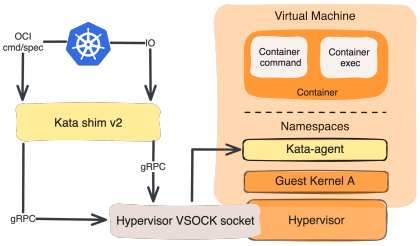
I. Motivation

Containers are lightweight, self-contained execution environments that encapsulate applications and their dependencies, providing a consistent and reproducible runtime environment. Achieving this versatility is made possible through a combination of operating system-level virtualization and resource isolation techniques. Unlike virtual machines, containers do not require a guest OS in each instance, resulting in smaller, faster, and highly portable units that can be executed everywhere. These are mainly the reasons all actors in the context of networking are shifting to the use of containers. In MLSysOps[1], we employ a pure cloud-native approach, across the continuum, compute and network. Figure 1(a) shows a high-level logical overview of the system components involved when two containers are running on a host system.

Limitations of containers: Despite their numerous advantages, containers have certain limitations, particularly in terms of security and isolation. Although containers provide a level of isolation by leveraging operating system features, they still share the underlying operating system kernel. This shared kernel introduces potential security risks, as a compromise within the kernel could impact the security and integrity of all containers running on the same host. Additionally, containers may not provide sufficient isolation for certain sensitive workloads or applications with strict security requirements. Furthermore, containers may face challenges when handling specific types of workloads, such as those with strict real-time requirements or resource-intensive applications that demand fine-grained control over hardware resources.



(a) Generic Containers (b) Sandboxed Containers



(c) Interaction with high-level container runtimes

Fig. 1. Process Isolation in generic and sandboxed containers

Sandboxing containers: To address the limitations of containers in terms of security, isolation [1] and resource control, container sandboxing comes into play. By encapsulating containers within microVMs, each with its dedicated kernel instance, stronger isolation and security are achieved. The use of microVMs ensures that any compromise within a specific microVM remains contained, mitigating potential security risks from the shared underlying kernel. With container sandboxing, containers can overcome their limitations in terms of security, isolation, and handling diverse workloads, making them more robust and suitable for a wide range of

[1] https://mlsysops.eu

applications across various domains. Figure 1(b) presents the high-level concept of container sandboxing using kata-containers.

Sandboxing in Edge Devices: Edge devices pose significant challenges in terms of limited computing resources and their heterogeneous nature. This requires developing specialized software that can seamlessly run across diverse edge devices, accommodating their unique characteristics. Developers must account for compatibility issues, adaptability, and the need for device-specific optimizations. Employing cloud-native techniques mitigate these issues. However, multi-tenancy adds another layer of complexity. When multiple deployments share the same edge devices, isolation becomes crucial to ensure the security and integrity of each application and its data. Therefore, sandboxing techniques, such as microVMs, can be employed to provide isolated execution environments for individual deployments, mitigating the risk of interference or unauthorized access.

II. Enhancing sandbox mechanisms

Adding a full virtualization stack (hypervisor, guest kernel, rootfs) to a simple network function incurs overhead [2] associated with the instantiation time, as well as the memory and CPU footprint of the actual workload on the compute resources. To alleviate this overhead, while enhancing isolation we choose to use unikernels [3]. Specifically, we introduce *urunc*[2], our own custom container runtime that is able to spawn unikernels. Coupled with the integration of *vAccel*[3] with kata-containers and urunc, we unify the deployment and execution paradigm of any network service function / application function across the whole MLSysOps continuum: Cloud-based and Edge-based. Users provide an application description, based on containerized workloads (OCI artifacts) and in each part of the continuum, based on underlying resources and hardware capabilities, the respective execution mode is selected and the workload is spawned either as a generic container, a sandboxed container or a unikernel. Taking advantage of vAccel (as an API-remoting hardware-acceleration framework) workloads enjoy the benefits of hardware-accelerated functions (eg. ML inference) on any part of the continuum, regardless of the underlying hardware-specific implementation.

We demonstrate the various execution modes using a simple serverless `httpreply` function[4]. Moreover, we demonstrate the hardware-acceleration abstraction using a simple ML model, running on two modes, CPU and GPU, using the same binary application (OCI artifact).

[2]`https://github.com/nubificus/urunc`
[3]`https://docs.vaccel.org`
[4]`https://github.com/nubificus/httpreply-go`

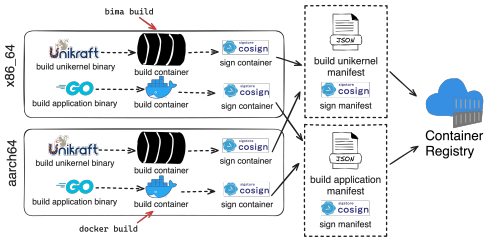


Fig. 2. Build OCI artifacts for deployment

Finally, we demonstrate the building process for the OCI artifacts we use (Figure 2), focusing on the interoperable OCI manifest creation (multiple architectures, multiple modes of execution).

III. Conclusions

This framework provides secure, efficient, and scalable deployment of applications in modern cloud environments, exemplifying the MLSysOps project's commitment to leveraging cutting-edge technologies for enhanced security and performance in cloud-native workloads (Figure 2).

References

[1] MITRE, "CVE-list related to containers.." CVE list of vulnerabilities related to the term 'containers': https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=containers, June 30 2024.

[2] V. van Rijn and J. S. Rellermeyer, "A fresh look at the architecture and performance of contemporary isolation platforms," in *Proceedings of the 22nd International Middleware Conference*, Middleware '21, (New York, NY, USA), p. 323–335, Association for Computing Machinery, 2021.

[3] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: library operating systems for the cloud," *SIGARCH Comput. Archit. News*, vol. 41, p. 461–472, mar 2013.